CMSC 330: Organization of Programming Languages

Ruby Regular Expressions
String Processing in Ruby

- Earlier, we motivated scripting languages using a popular application of them: string processing.
- The Ruby **String** class provides many useful methods for manipulating strings:
  - Concatenating them, grabbing substrings, searching in them, etc.
- A key feature in Ruby is its native support for regular expressions:
  - Very useful for parsing and searching
  - First gained popularity in Perl
String Operations in Ruby

• "hello".index("l", 0)
  ➢ Return index of the first occurrence of string in s, starting at n
• "hello".sub("h", "j")
  ➢ Replace first occurrence of "h" by "j" in string
  ➢ Use gsub ("global" sub) to replace all occurrences

Consider these three examples again

• All involve searching in a string for a certain pattern
• What if we want to find more complicated patterns?
  ➢ Find Steve, Stephen, Steven, Stefan, Esteve
Regular Expressions

- A way of describing patterns or sets of strings
  - Searching and matching
  - Formally describing strings
    - The symbols (lexemes or tokens) that make up a language

- Common to lots of languages and tools
  - awk, sed, perl, grep, Java, OCaml, C libraries, etc.
    - Popularized (and made fast) as a language feature in Perl

- Based on some really elegant theory
  - Future lecture
Example Regular Expressions in Ruby

- `/Ruby/`
  - Matches exactly the string "Ruby"
  - Regular expressions can be delimited by /’s
  - Use \ to escape /’s in regular expressions

- `/Ruby|OCaml|Java)/`
  - Matches either "Ruby", "OCaml", or "Java"

- `/Ruby|Regular)/ or /R(uby|egular)/`
  - Matches either "Ruby" or "Regular"
  - Use ( )’s for grouping; use \ to escape ( )’s
Using Regular Expressions

- Regular expressions are instances of `Regexp`
  - We’ll see use of a `Regexp.new` later

- Basic matching using `=~` method of `String`

  ```ruby
  line = gets               # read line from standard input
  if line =~ /Ruby/ then    # returns nil if not found
    puts "Found Ruby"
  end
  ```

- Can use regular expressions in index, search, etc.

  ```ruby
  offset = line.index(/(MAX|MIN)/)  # search starting from 0
  line.sub(/(Perl|Python)/, "Ruby")  # replace
  line.split(/(\t|\n| )/)           # split at tab, space, newline
  ```
Repetition in Regular Expressions

- /Ruby*/
  - {"", "Ruby", "RubyRuby", "RubyRubyRuby", ...}
  - * means **zero or more occurrences**

- /Ruby+/  
  - {"Ruby", "Rubyy", "Rubyyy", ...}
  - + means **one or more occurrence**
  - so /e+/ is the same as /ee*/

- /(Ruby)?/
  - {"", "Ruby"}
  - ? means **optional**, i.e., zero or one occurrence
Repetition in Regular Expressions

- /(Ruby){3}/
  - {“RubyRubyRuby”}
  - {x} means repeat the search for exactly x occurrences

- /(Ruby){3,}/
  - {“RubyRubyRuby”, “RubyRubyRubyRubyRuby”, …}
  - {x,} means repeat the search for at least x occurrences

- /(Ruby){3, 5}/
  - {“RubyRubyRuby”, “RubyRubyRubyRubyRuby”, “RubyRubyRubyRubyRubyRuby”}
  - {x, y} means repeat the search for at least x occurrences and at most y occurrences
Watch Out for Precedence

- `/\(Ruby\)*/` means `{"", "Ruby", "RubyRuby", ...}
- `/Ruby*/` means `"Rub", "Ruby", "Rubyy", ...`

In general
- `\* {n}` and `+` bind most tightly
- Then concatenation (adjacency of regular expressions)
- Then `|`

Best to use parentheses to disambiguate
- Note that parentheses have another use, to extract matches, as we’ll see later
Character Classes

- `/[abcd]/`
  - \{"a", "b", "c", "d"\} (Can you write this another way?)

- `/[a-zA-Z0-9]/`
  - Any upper or lower case letter or digit

- `/[^0-9]/`
  - Any character except 0-9 (the ^ is like not and must come first)

- `/[\t\n ]/`
  - Tab, newline or space

- `/[a-zA-Z_\$][a-zA-Z_\$0-9]*/`
  - Java identifiers ($ escaped...see next slide)
## Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>any character</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>beginning of line</td>
<td>Using /^pattern$/ ensures entire string/line must match pattern</td>
</tr>
<tr>
<td>$</td>
<td>end of line</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>just a $</td>
<td></td>
</tr>
<tr>
<td>\d</td>
<td>digit, [0-9]</td>
<td></td>
</tr>
<tr>
<td>\s</td>
<td>whitespace, [t\r\n\f\s]</td>
<td></td>
</tr>
<tr>
<td>\w</td>
<td>word character, [A-Za-z0-9_]</td>
<td></td>
</tr>
<tr>
<td>\D</td>
<td>non-digit, [^0-9]</td>
<td></td>
</tr>
<tr>
<td>\S</td>
<td>non-space, [^t\r\n\f\s]</td>
<td></td>
</tr>
<tr>
<td>\W</td>
<td>non-word, [^A-Za-z0-9_]</td>
<td></td>
</tr>
</tbody>
</table>
Potential Character Class Confusions

- \(^\)
  - Inside character classes: *not*
  - Outside character classes: beginning of line

- [ ]
  - Inside regular expressions: character class
  - Outside regular expressions: array
    - Note: \([a-z]\) does not make a valid array

- ( )
  - Inside character classes: literal characters ( )
    - Note: \((0..2)\) does not mean 012
  - Outside character classes: used for grouping

- –
  - Inside character classes: range (e.g., a to z given by \([a-z]\))
  - Outside character classes: subtraction
Summary

- Let \( re \) represents an arbitrary pattern; then:
  - \( /re/ \) – matches regexp \( re \)
  - \( /(re_1|re_2)/ \) – match either \( re_1 \) or \( re_2 \)
  - \( /(re)^*/ \) – match 0 or more occurrences of \( re \)
  - \( /(re)^+/ \) – match 1 or more occurrences of \( re \)
  - \( /(re)?/ \) – match 0 or 1 occurrences of \( re \)
  - \( /(re)\{2\}/ \) – match exactly two occurrences of \( re \)
  - \( /[a-z]/ \) – same as (a|b|c|...|z)
  - \( /[^0-9]/ \) – match any character that is not 0, 1, etc.
  - \( ^, \$/ \) – match start or end of string
Try out regexps at rubular.com
Regular Expression Practice

Contains 3 b's, may not be consecutive.

/^[^b]*b[^b]*b[^b]*b[^b]*$/
Regular Expression Practice

- Starts with c, followed by one vowel, and any number of letters

/^c[aouei][a-z]*$/
Regular Expression Practice

- All letters are in alphabetic order

```
/^a*b*c*d*e*f*g*h*i*j*k*l*m*n*o*p*r*t*$/
```
Regular Expression Practice

- Contains sss or ccc

```
/s{3}|c{3}/
```
Regular Expression Practice

- contains 2 ab

  /\(ab\)\{2\}/

- contains 2 b

  /b\{2\}/
Regular Expression Practice

- Starts with a, 0 or 1 letter after that

/\^a[a-z]?$/
Regular Expression Practice

- Contains one or more ab or ba

/(ab|ba)+/
Regular Expression Practice

- steve, steven, or stephen

/^ste(ve|phen|ven)$/
Regular Expression Practice

- Even length string

/^(.\.)*$/
Regular Expression Practice

- Even number of vowels

/^(^[^aouei]*[^aouei][^aouei]*[^aouei][^aouei]*[^aouei])*$/
Regular Expression Practice

- starts with not-b, 0 or more times of any number of b followed by one not-b

```
/^[^b]+(b+[^[b])*/
```
Regular Expression Practice

- Make Ruby regular expressions representing:
  - All lines beginning with a or b: `/^\(a|b\)\$/`
  - All lines containing at least two (only alphabetic) words separated by white-space: `/[a-zA-Z]+\s+[a-zA-Z]+\$/`
  - All lines where a and b alternate and appear at least once: `/^\((ab)+ a?\) | ((ba)+ b?)$/`
  - An expression which would match both of these lines (but not radically different ones):
    - CMSC330: Organization of Programming Languages: Fall 2018
    - CMSC351: Algorithms: Fall 2018
Quiz 1

How many different strings could this regex match?

/\^Hello, Anyone awake?$/

A. 1
B. 2
C. 4
D. More than 4
Quiz 1

How many different strings could this regex match?

/\^Hello, Anyone awake?$/

A. 1
B. 2
C. 4
D. More than 4
Quiz 2

Which regex is not equivalent to the others?

A. ^[crab]$  
B. ^c?r?a?b?$  
C. ^(c|r|a|b)$  
D. ^([cr]|[ab])$
Quiz 2

Which regex is not equivalent to the others?

A. `^[crab]$`
B. `^c?r?a?b?$`
C. `^(c|r|a|b)$`
D. `^([cr]|[ab])$`
Quiz 3

Which string does not match the regex?

\/[a-z]{4}\d{3}/

A. “cmsc\d\d\d”
B. “cmsc330”
C. “hellocmsc330”
D. “cmsc330world”
Quiz 3

Which string does not match the regex?

Recall that without ^ and $, a regex will match any substring

\/[a-z]{4}\d{3}/

A. “cmsg\d\d\d”
B. “cmisc330”
C. “hellocmisc330”
D. “cmisc330world”
Extracting Substrings based on R.E.’s Method 1: Back References

Two options to extract substrings based on R.E.’s:

- Use back references
  - Ruby remembers which strings matched the parenthesized parts of r.e.’s
  - These parts can be referred to using special variables called back references (named $1, $2,...)
Back Reference Example

gets =~ /^Min: (\d+) Max: (\d+)/
min, max = $1, $2
puts "mini=#{min} maxi=#{max}"

Input
Min: 1 Max: 27
Min: 10 Max: 30
Min: 11 Max: 30
Min: a Max: 24

Output
mini=1 maxi=27
mini=10 maxi=30
mini= maxi=
mini= maxi=

Extra space messes up match
Not a digit; messes up match
Back References are Local

- Warning
  - Despite their names, $1 etc are local variables
  - (Normally, variables starting with $ are global)

```ruby
def m(s)
  s =~ /(Foo)/
  puts $1  # prints Foo
end
m("Foo")
puts $1    # prints nil
```
## Back References are Reset

### Warning 2
- If another search is performed, all back references are reset to nil

<table>
<thead>
<tr>
<th>gets =~ /(h)e(ll)o/</th>
<th>hello</th>
</tr>
</thead>
<tbody>
<tr>
<td>puts $1</td>
<td>h</td>
</tr>
<tr>
<td>puts $2</td>
<td>ll</td>
</tr>
<tr>
<td>gets =~ /h(e)llo/</td>
<td>hello</td>
</tr>
<tr>
<td>puts $1</td>
<td>e</td>
</tr>
<tr>
<td>puts $2</td>
<td>nil</td>
</tr>
<tr>
<td>gets =~ /hello/</td>
<td>hello</td>
</tr>
<tr>
<td>puts $1</td>
<td>nil</td>
</tr>
</tbody>
</table>
Quiz 4

What is the output of the following code?

```ruby
s = "help I’m stuck in a text editor"
s =~ /[A-Z]+/;
puts $1
```

A. help
B. I
C. I’m
D. I’m stuck in a text editor
What is the output of the following code?

```ruby
s = "help I'm stuck in a text editor"
s =~ /[A-Z]+/;
puts $1
```

A. help
B. I
C. I’m
D. I’m stuck in a text editor
Quiz 5

What is the output of the following code?

“Why was 6 afraid of 7?” =~ /\d\s(\w+).*(\d)/
puts $2

A. afraid
B. Why
C. 6
D. 7
Quiz 5

What is the output of the following code?

“Why was 6 afraid of 7?” =~ /\d\s(\w+).*\d/puts $2

A. afraid
B. Why
C. 6
D. 7
Method 2: String.scan

- Also extracts substrings based on regular expressions
- Can optionally use parentheses in regular expression to affect how the extraction is done
- Has two forms that differ in what Ruby does with the matched substrings
  - The first form returns an array
  - The second form uses a code block
    - We’ll see this later
First Form of the Scan Method

- `str.scan(regexp)`
  - If `regexp` doesn't contain any parenthesized subparts, returns an array of matches
    - An array of all the substrings of `str` which matched
  - Note: these strings are chosen sequentially from as yet unmatched portions of the string, so while “330 Fall” does match the regular expression above, it is not returned since “330” has already been matched by a previous substring.

```ruby
s = "CMSC 330 Fall 2018"
s.scan(/\S+ \S+/)
# returns array ["CMSC 330", "Fall 2018"]

s.scan(/\S{2}/)
# => ["CM", "SC", "33", "Fa", "11", "20", "18"]
```
First Form of the Scan Method (cont.)

- If `regexp` contains parenthesized subparts, returns an array of arrays
  - Each sub-array contains the parts of the string which matched one occurrence of the search
  ```ruby
  s = "CMSC 330 Fall 2018"
  s.scan(/(\S+) (\S+)/)  # [['CMSC', '330'],
  # ['Fall', '2018']]
  ```
  - Each sub-array has the same number of entries as the number of parenthesized subparts
  - All strings that matched the first part of the search (or $1 in back-reference terms) are located in the first position of each sub-array
Practice with Scan and Back-references

> ls -l

drwx------ 2 sorelle sorelle 4096 Feb 18 18:05 bin
-rw-------- 1 sorelle sorelle 674 Jun 1 15:27 calendar
drwx------ 3 sorelle sorelle 4096 May 11 2006 cmsc311
drwx------ 2 sorelle sorelle 4096 Jun 4 17:31 cmsc330
drwx------ 1 sorelle sorelle 4096 May 30 19:19 cmsc630
drwx------ 1 sorelle sorelle 4096 May 30 19:20 cmsc631

Extract just the file or directory name from a line using

- **scan**
  
  ```
  name = line.scan(/\S+$/)  # ["bin"]
  ```

- **back-references**
  
  ```
  if line =~ /\S+$)/
    name = $1  # "bin"
  end
  ```
Quiz 6

What is the output of the following code?

```ruby
s = "Hello World"
t = s.scan(/\w{2}/).length
puts t
```

A. 3
B. 4
C. 5
D. 6
Quiz 6

What is the output of the following code?

```ruby
s = "Hello World"
t = s.scan(/\w{2}/).length
puts t
```

A. 3
B. 4
C. 5
D. 6
Quiz 7

What is the output of the following code?

```
s = "To be, or not to be!"
a = s.scan(/((\S+) (\S+)/)
puts a.inspect
```

A. ["To", "be,", "or", "not", "to", "be!"]
B. [["To", "be,"], ["or", "not"], ["to", "be!"]]
C. ["To", "be,"]
D. ["to", "be!"]
Quiz 7

What is the output of the following code?

```ruby
s = "To be, or not to be!"
a = s.scan(/(\S+) (\S+)/)
puts a.inspect
```

A. ["To","be","or","not","to","be!" ]
B. [["To","be,"],["or","not"],["to","be!" ]]
C. ["To","be,"]
D. ["to","be!" ]
Second Form of the Scan Method

- Can take a code block as an optional argument

- `str.scan(regexp) { |match| block }`
  - Applies the code block to each match
  - Short for `str.scan(regexp).each { |match| block }
  - The regular expression can also contain parenthesized subparts
Example of Second Form of Scan

Sums up three columns of numbers

input file:
will be read line by line, but
column summation is desired

12 34 23
19 77 87
11 98 3
2 45 0

sum_a = sum_b = sum_c = 0
while (line = gets)
    line.scan(/((d+)\s+(d+)\s+(d+)/) { |a,b,c|
        sum_a += a.to_i
        sum_b += b.to_i
        sum_c += c.to_i
    }
end
printf("Total: %d %d %d\n", sum_a, sum_b, sum_c)

converts the string
to an integer
Practice: Amino Acid counting in DNA

Write a function that will take a filename and read through that file counting the number of times each group of three letters appears so these numbers can be accessed from a hash.

(assume: the number of chars per line is a multiple of 3)

gcggcattcagcaccctgcttaagcaatccagatgaatgtgtataacatataccggtccatactgaagcattcattgaggctagcgcctgataaacagtacgcgctaacaatgggggttggcaatacggtgcgattactaagagccggaaccacacacacccgtaaggatgagacgtggttaacataataatcctcgttcaagcagtggcagatgtttcagtaaggaataggtggggcctactacccatgtacctacataacatatgtgtacataaacagagatcgtcagaatctttgagacggtcaatgggtacgagactatatactcaactccggacgtatgctgtctactgggtcaacctcgttactgccgga
def countaa(filename):
    file = File.new(filename, "r")
    lines = file.readlines
    hash = Hash.new
    lines.each{ |line|
        acids = line.scan(/.../)
        acids.each{ |aa|
            if hash[aa] == nil
                hash[aa] = 1
            else
                hash[aa] += 1
            end
        }
    }
end